Fixed-Parameter Algorithms, IA166

Sebastian Ordyniak

Faculty of Informatics
Masaryk University Brno

Spring Semester 2013

Outline

Color Coding

Motivation

- works best when we need to ensure that a small number of "things" are disjoint.
- We demonstrate it on the problem of finding s-t path of length exactly k.
- Randomized algorithm, but can be derandomized using standard techniques.
- Very robust technique, we can use it as an "opening step" when investigating a new problem.

Outline

Introduction

k-s-t-PATH **Parameter: k**

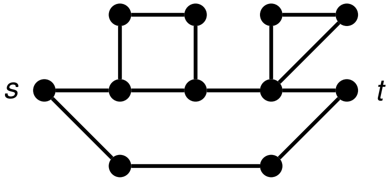
Input: Graph G, 2 vertices s and t, and a natural number k.
Question: Find an s-t-path, i.e. a path from s to t in G, with exactly k internal vertices.

Remark

The problem is NP-hard because it contains the s-t-HAMILTONIAN PATH problem.

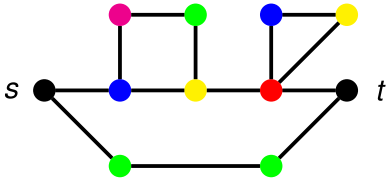
Basic Idea

- Assign k colors to the vertices in $V(G) \setminus \{s, t\}$ uniformly and independently at random.
- Check if there is a colorful s-t-path, i.e., a path where each color appears exactly once on the internal vertices. If so output YES, if not output NO.



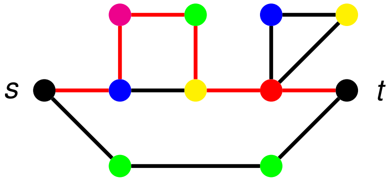
Basic Idea

- Assign k colors to the vertices in $V(G) \setminus \{s, t\}$ uniformly and independently at random.
- Check if there is a colorful s-t-path, i.e., a path where each color appears exactly once on the internal vertices. If so output YES, if not output NO.



Basic Idea

- Assign k colors to the vertices in $V(G) \setminus \{s, t\}$ uniformly and independently at random.
- Check if there is a colorful s-t-path, i.e., a path where each color appears exactly once on the internal vertices. If so output YES, if not output NO.



Basic Idea

This gives us a randomized algrithm for k-s-t-PATH such that:

- Given a NO instance of k-s-t-PATH, the algorithm always outputs NO.
- Given a YES instance of k-s-t-PATH, the algorithm outputs YES with probability:

$$\frac{k!}{k^k} \approx \sqrt{2\pi k}\left(\frac{1}{e}\right)^k > e^{-k}$$

Here we use Stirling's formula: $k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k$.

Basic Idea

Observation

Let A be a randomized algorithm with success rate at least p. Then repeating A at least $1/p$-times leads to an error probability of at most $(1-p)^{1/p} \leq (e^{-p})^{1/p} = e^{-1} = 1/e \approx 0.38$ (Using the fact that $1 - x \leq e^{-x}$).

- Hence, if $p > e^{-k}$ then the error probability of A is at most $1/e$ after e^k repetitions.
- Repeating the algorithm ce^k times (for some constant c) decreases the error probability of the algorithm to an arbitrary small constant, e.g., by trying $100e^k$ random colorings, the error probability becomes e^{-100}.

A Monte Carlo FPT-algorithm for k-s-t-PATH

Provided that we can find a colorful s-t-Path in time $f(k)n^c$ the above randomized algorithm decides k-s-t-PATH with arbitrary low error probability in time $O^*(e^k f(k)n^c)$. Such a randomized algorithm is also called a Monte Carlo algorithm.

There are 2 important questions remaining:

Question (1)

How to find a colorful s-t-path in polynomial time?

Question (2)

Is it possible to derandomize the above algorithm?

Finding a Colorful s-t-Path

k-COLORFUL PATH **Parameter: k**

Input: A graph G, 2 vertices s and t of G, and a vertex-coloring c of G with k colors.
Question: Does G contain a colorful s-t-Path?

We will now show two methods to solve the above problem:

- Method 1: Trying all permutations;
- Method 2: Dynamic Programming.

Method 1: Trying all permutations

The colors encountered on a colorful s-t-path form a permutation π of $\{1, \ldots, k\}$.

We try all $k!$ permutations. For a fixed permutation it is easy to check if there is a path with this order of colors.

Method 1: Trying all permutations

Let π be such a permutation. The following algorithm decides whether G has a colorful s-t-path representing π:

- Remove edges connecting vertices colored by non-neighboring colors with respect to π.
- Direct the remaining edges according to π.
- Check whether there is a directed s-t-path.
- Running time is $O(|E(G)|)$.

Method 1: Trying all permutations

Theorem

k-COLORFUL PATH can be decided in time $O(k!|E(G)|)$, for an instance (G, c, k).

Corollary

k-s-t-PATH can be decided by a randomized algorithm with arbitrary high constant success probability in time $O(e^k k!|E(G)|)$.

Method 2: Dynamic Programming

We introduce $2^k|V(G)|$ boolean variables, i.e., for every $v \in V(G)$ and every $C \subseteq \{1, \ldots, k\}$ we introduce a variable $x(v, C)$ that is TRUE iff G contains an s-v-path that contains only colors in C and each color in C appears exactly once on this path.

Method 2: Dynamic Programming

- Clearly, $x(s, \emptyset) = $ TRUE. Furthermore, we can use the following recurrence for a vertex v with color r:

$$x(v, C) = \bigvee_{\{u,v\} \in E(G)} x(u, C \setminus \{r\})$$

- Using the above recurrence we can determine the values of every $x(v, C)$ from the values of every $x(v, C')$ with $|C'| = |C| - 1$. This allows us to determine the values of all these variables in time $O(2^k|E(G)|)$.
- Clearly, G has a colorful s-t-path iff $x(v, \{1, \ldots, k\}) = $ TRUE for some neighbor v of t.

Method 2: Dynamic Programming

Theorem

k-COLORFUL PATH can be decided in time $O(2^k|E(G)|)$, for an instance (G, c, k).

Corollary

k-s-t-PATH can be decided by a randomized algorithm with arbitrary high constant success probability in time $O((2e)^k|E(G)|)$.

Derandomization

Using Method 2, we obtain a $O((2e)^k|E(G)|)$ time Monte Carlo algorithm. How can we make it deterministic?

Definition

A family $\mathcal{H}$ of functions from $\{1,\ldots,n\}$ to $\{1,\ldots,k\}$ is a *k-perfect* family of hash functions if for every $S \subseteq \{1,\ldots,n\}$ with $|S| = k$, there is a $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for every $x, y \in S$ with $x \neq y$.

Instead of trying $O(e^k)$ random colorings, we go through a k-perfect family $\mathcal{H}$ of hash functions. If there is a solution then the internal vertices are colorful for at least 1 such function and our algorithm returns YES.

Derandomization

Theorem

There is a k-perfect family of hash functions from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$ having size at most $2^{O(k)} \log n$ and such a family can be constructed in polynomial time with respect to the size of the family.

Corollary

There is a deterministic $O(2^{O(k)} n^{O(1)})$ time algorithm for the k-s-t-PATH problem.

Some Simple Generalizations

k-CYCLE **Parameter: k**

Input: Graph G and a natural number k.
Question: Does G contain a cycle of length exactly k.

By computing k-s-t-PATH for every pair of distinct and adjacent vertices s and t of G we obtain:

Corollary

There is a deterministic $O(2^{O(k)}n^{O(1)})$ time algorithm for the k-CYCLE problem.

Some Simple Generalizations

k-LONGEST PATH **Parameter: k**

Input: Graph G and a natural number k.
Question: Does G contain a path of length at least k.

By computing k-s-t-PATH for every pair of distinct vertices s and t of G and observing that G contains a path of lenght at least k iff G contains a path of length exactly k we obtain:

Corollary

There is a deterministic $O(2^{O(k)}n^{O(1)})$ time algorithm for the k-LONGEST PATH problem.

Outline

Introduction

k-TREE SUBGRAPH **Parameter: $|V(T)|$**

Input: A tree T and a graph G.
Question: Does G contain T as a subgraph?

As before, we start by solving its colorful version:

k-COLORFUL-TREE SUBGRAPH **Parameter: $|V(T)|$**

Input: A tree T and a graph G with a $|V(T)|$-vertex coloring $c : V(G) \to |V(T)|$.
Question: Does G contain a colorful copy of T as a subgraph?

A Dynamic Programming Approach

W.l.o.g. we can assume that the tree T is rooted at some arbitrary vertex $r \in V(T)$. We denote by $T(t)$ the subtree of T rooted in t.

We solve k-COLORFUL-TREE SUBGRAPH via a dynamic programming algorithm that computes a set of records in a bottom-up manner along the tree T, i.e, starting from the leaves of T and progressing to the root of T. For every tree node t of T the set of records (denoted $\mathcal{R}(t)$) contains all pairs (v, C) such that $v \in V(G)$, $C \subseteq \{1, \ldots, k\}$ and G contains a colorful copy (with respect to C) of $T(t)$ where v takes the role of t.

A Dynamic Programming Approach

Recursion Start:

If $l \in V(T)$ is a leave of T, then $\mathcal{R}(l) := \{\,(v, \{c(v)\}) : v \in V(G)\,\}$. Hence, $\mathcal{R}(l)$ can be computed in time $O(|V(G)|)$ for every leave node l of T.

A Dynamic Programming Approach

Recursion Step:

If t is an inner node of T with children $t_1, \ldots, t_l$, then

$$\mathcal{R}(t) := \{ (v, C) : v \in V(G) \text{ and there is an ordered partition } (C_1, \ldots, C_l) \text{ of } C \setminus c(v) \text{ and neighbors } v_1, \ldots, v_l \text{ of } v \text{ in } G \text{ such that: } (v_i, C_i) \in \mathcal{R}(t_i) \}$$

Question

How can we compute the above efficiently?

A Dynamic Programming Approach

Lemma

If t is an inner node of T with children $t_1, \dots, t_l$ and let $v \in V(G)$ with neighbors $n_1, \dots, n_r$ in G. Then $(v, C) \in \mathcal{R}(t)$ iff $c(v) \in C$ and there is an ordered partition $(C_1, \dots, C_l)$ of $C \setminus \{c(v)\}$ such that the bipartite graph $B(t)$ with vertices

$$\{\, t_1, \dots, t_l \,\} \cup \{\, n_1, \dots, n_r \,\}$$

and edges

$$\{\, \{t_i, n_j\} : (n_j, C_i) \in \mathcal{R}(t_i) \,\}$$

has a matching that saturates $\{t_1, \dots, t_l\}$.

A Dynamic Programming Approach

Because of the above Lemma we can decide whether a potential record (v, C) is in the set $\mathcal{R}(t)$ as follows:

(1) Guess an ordered partition $(C_1, \ldots, C_l)$ of $C \setminus \{c(v)\}$.

(2) Construct the bipartite graph $B(t)$ as in the above lemma

(4) Check whether $B(t)$ has a perfect matching. If so output YES, otherwise output NO.

A Dynamic Programming Approach

Because of the above Lemma we can decide whether a potential record (v, C) is in the set $\mathcal{R}(t)$ as follows:

(1) Guess an ordered partition $(C_1, \ldots, C_l)$ of $C \setminus \{c(v)\}$. This takes time $O(2^l l!) = O(2^{|V(T)|}(|V(T)|)!)$.

(2) Construct the bipartite graph $B(t)$ as in the above lemma This takes time $O(lr) = O(|V(T)||V(G)|)$.

(4) Check whether $B(t)$ has a perfect matching. If so output YES, otherwise output NO. This takes time $O((lr) = O(|V(T)||V(G)|)$.

A Dynamic Programming Approach

Because of the above Lemma we can decide whether a potential record (v, C) is in the set $\mathcal{R}(t)$ as follows:

(1) Guess an ordered partition $(C_1, \dots, C_l)$ of $C \setminus \{c(v)\}$. This takes time $O(2^l l!) = O(2^{|V(T)|}(|V(T)|)!)$.

(2) Construct the bipartite graph $B(t)$ as in the above lemma This takes time $O(lr) = O(|V(T)||V(G)|)$.

(4) Check whether $B(t)$ has a perfect matching. If so output YES, otherwise output NO. This takes time $O((lr) = O(|V(T)||V(G)|)$.

Hence, we can decide whether $(v, C) \in \mathcal{R}(t)$ in time $O(2^l l! lr)$ or equivalently in time $O(2^{|V(T)|}(|V(T)|!)|V(T)||V(G)|)$.

A Dynamic Programming Approach

Since there are at most $O(2^{|V(T)|}|V(G)|)$ potential records for every tree node and at most $|V(T)|$ tree nodes we obtain the following:

Theorem

k-COLORFUL-TREE SUBGRAPH can be decided in time $O(4^{|V(T)|}(|V(T)|!)(|V(T)|)^2(|V(G)|)^2)$.

By running the above algorithm for every hash function of a perfect family of hash functions, we obtain:

Corollary

k-TREE SUBGRAPH can be decided in time $O(2^{O(|V(T)|)}(|V(T)|!)(|V(T)|)^2(|V(G)|)^2)$.

Even More General

Let $\mathcal{C}$ be an arbitrary class of graphs.

k-$\mathcal{C}$-SUBGRAPH **Parameter:** $|V(H)|$

Input: A graph $H \in \mathcal{C}$ and a graph G.
Question: Does G contain H as a subgraph?

- Using the above algorithms we have seen that k-$\mathcal{C}$-SUBGRAPH is FPT if $\mathcal{C}$ is the class of all trees respectively cycles.
- Because k-$\mathcal{C}$-SUBGRAPH is equivalent to the k-CLIQUE problem if $\mathcal{C}$ is the class of all cliques we can note hope for an FPT algorithm in general (unless FPT=W[1]).

Even More General

Question

Is there some class $\mathcal{C}$ in between trees (cycles) and cliques that allows for fixed-parameter tractability of k-$\mathcal{C}$-SUBGRAPH?

Outline

Introduction

Theorem

Let $\mathcal{C}$ be a class of graphs of treewidth at most w. Then k-$\mathcal{C}$-SUBGRAPH can be solved in time $O(2^{O(|V(H)|)}(|V(G)|)^{w+O(1)})$ (if a tree decomposition of the graph H of width w is given as the input).

Corollary

Let $\mathcal{C}$ be a class of graphs of bounded treewidth. Then k-$\mathcal{C}$-SUBGRAPH is fixed-parameter tractable.

Solving the Colorful Problem

We first need to solve the following problem:

k-$\mathcal{C}$-COLORFUL SUBGRAPH **Parameter: $|V(H)|$**

Input: A graph $H \in \mathcal{C}$ and a graph G with a $|V(H)|$-vertex coloring $c : V(G) \to \{1, \dots, |V(H)|\}$.
Question: Does G contain a colorful subgraph isomorphic to H?

Solving the Colorful Problem

Let (T, X) be a nice tree decomposition of H and let $t \in V(T)$. As always we compute a set of records $\mathcal{R}(t)$ for every $t \in V(T)$ in a bottom up manner. This time a record is a pair (ϕ, C) such that ϕ is a 1-to-1 mapping between vertices in $X(t)$ and exaclt $|X(t)|$ vertices in $V(G)$ and $C \subseteq \{1, \ldots, |V(H)|\}$ is a set of colors.

The semantics of a record is as follows:

$(\phi, C) \in \mathcal{R}(t)$ iff G contains a colorful copy of $X(t)$ using every color in C exactly once such that the vertex $\phi(v)$ is mapped to v for every $v \in X(t)$.

Clearly, the solution for the k-$\mathcal{C}$-Colorful Subgraph problem can be easily obtained from $\mathcal{R}(r)$ by checking wether $(\emptyset, \{1, \ldots, |V(H)|\}) \in \mathcal{R}(r)$.

Solving the Colorful Problem

Let (T, X) be a nice tree decomposition of H and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{\, ((v \to v'), \{c(v')\}) : v' \in V(G)\,\}$.

Solving the Colorful Problem

Let (T, X) be a nice tree decomposition of H and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{\, ((v \to v'), \{c(v')\}) : v' \in V(G)\,\}$.

Can be computed in time $|V(G)|$.

Solving the Colorful Problem

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\begin{aligned}\mathcal{R}(t) := \{\, (\phi + (v \to v'), C \cup \{c(v')\}) :\; & (\phi, C) \in \mathcal{R}(t') \text{ and } v' \in V(G) \text{ and} \\ & \forall_{w \in X(t')} \{v, w\} \in E(H) \to \{v', \phi(w)\} \in E(G) \,\}\end{aligned}$$

Solving the Colorful Problem

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\begin{aligned}\mathcal{R}(t) := \{\, &(\phi + (v \to v'), C \cup \{c(v')\}) : \\ &(\phi, C) \in \mathcal{R}(t') \text{ and } v' \in V(G) \text{ and} \\ &\forall_{w \in X(t')} \{v, w\} \in E(H) \to \{v', \phi(w)\} \in E(G) \,\}\end{aligned}$$

Can be computed in time $O(|\mathcal{R}(t')||X(t)||V(G)|)$.

Solving the Colorful Problem

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, (\phi[X(t)], C) : (\phi, C) \in \mathcal{R}(t') \,\}$$

Solving the Colorful Problem

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, (\phi[X(t)], C) : (\phi, C) \in \mathcal{R}(t')\,\}$$

Can be computed in time $O(|\mathcal{R}(t')|)$.

Solving the Colorful Problem

t is a join node with children t_1 and t_2

$$\begin{aligned}\mathcal{R}(t) := \{\, (\phi_1, C_1 \cup C_2) :\ & \\ & (\phi_1, C_1) \in \mathcal{R}(t_1) \text{ and } (\phi_2, C_2) \in \mathcal{R}(t_2) \text{ and} \\ & \phi_1 == \phi_2 \text{ and } C_1 \cap C_2 = \{\, c(\phi_1(v)) : v \in X(t) \,\}\end{aligned}$$

Solving the Colorful Problem

t is a join node with children t_1 and t_2

$$\begin{aligned}\mathcal{R}(t) := \{\, (\phi_1, C_1 \cup C_2) :\ & (\phi_1, C_1) \in \mathcal{R}(t_1) \text{ and } (\phi_2, C_2) \in \mathcal{R}(t_2) \text{ and} \\ & \phi_1 == \phi_2 \text{ and } C_1 \cap C_2 = \{\, c(\phi_1(v)) : v \in X(t) \,\} \,\}\end{aligned}$$

Can be computed in time $O(\max\{|\mathcal{R}(t_1)|, |\mathcal{R}(t_2)|\})$.

Solving the Colorful Problem

Because there are at most $O(|V(H)|)$ tree nodes and for each tree node we need time at most $O(|\mathcal{R}(t)||X(t)||V(G)|)$ the total running time of the algorithm is $O(|\mathcal{R}(t)||X(t)||V(H)||V(G)|)$ or equivalently $O(2^{|V(H)|}(|V(G)|^{w+1}(w+1)|V(H)||V(G)|)$.

Theorem

Let $\mathcal{C}$ be a class of graphs of treewidth at most w. Then k-$\mathcal{C}$-Colorful Subgraph can be decided in time $O(2^{|V(H)|}(|V(G)|)^{w+2}(w+1)|V(H)|)$ (if a tree decomposition of the graph H of width w is given as the input).

Solving the whole problem

Theorem

Let $\mathcal{C}$ be a class of graphs of treewidth at most w. Then k-$\mathcal{C}$-COLORFUL SUBGRAPH can be decided in time $O(2^{|V(H)|}(|V(G)|)^{w+2}(w+1)|V(H)|)$ (if a tree decomposition of the graph H of width w is given as the input).

Using perfect hash functions we obtain:

Corollary

Let $\mathcal{C}$ be a class of graphs of treewidth at most w. Then k-$\mathcal{C}$-SUBGRAPH can be decided in time $O(2^{O(|V(H)|)}(|V(G)|)^{w+O(1)})$ (if a tree decomposition of the graph H of width w is given as the input).

Outline

Color Coding – Summary

- Color Coding is a useful technique for solving various subgraph problems; fixing a coloring makes dynamic programming possible.
- This method can be generalized to finding embendings between general relations structures.
- Color Coding makes use of the fact that we can guess properties of a solution.
- Giving a randomized (Monto Carlo) algorithm (as a first step) is often easier than giving an algorithm that is always correct.